AFRL-IF-RS-TR-2006-288
Final Technical Report
September 2006

AUTOMATIC DETECTION OF ANOMALOUS BEHAVIOR IN NETWORKS

Purdue University

STINFO FINAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

FOR THE DIRECTOR:

/s/	/s/
ANDREW J. KARAM	WARREN H. DEBANY, JR.
Work Unit Manager	Technical Advisor, Information Grid Division
	Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* SEP 2006	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* Sep 02 – Mar 06

4. TITLE AND SUBTITLE AUTOMATIC DETECTION OF ANOMALOUS BEHAVIOR IN NETWORKS	**5a. CONTRACT NUMBER** F30602-02-2-0217
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 33140F
6. AUTHOR(S) C. E. Brodley	**5d. PROJECT NUMBER** 7820
	5e. TASK NUMBER 96
	5f. WORK UNIT NUMBER 10
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Purdue University 1063 Hovde Hall West Lafayette Indiana 47906	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFGB 525 Brooks Rd Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2006-288

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA#06-641

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Detection of anomalous behavior in networks is a difficult problem. We created automatic tools that will detect, through traffic monitoring, anomalous behaviors in computer networks. Because signature techniques cannot detect new forms of attacks, we focused on designing adaptive solution to quickly detect new (and old) attacks while minimizing the false alarm rate. Our approach is to form a model of the normal behavior of a network element and then monitor incoming/outing traffic for anomalies. As part of our research, we have also researched methods to model human behavior to detect anomalies in user patterns through mouse movements. In particular, we monitor each user’s keystroke, mouse and GUI behavior to determine if he/she is a valid user or an imposter.

15. SUBJECT TERMS
Anomaly Detection, DDOS, NDOS, Detection, “Characterization of Normal Behavior,” Network Servers, Monitor, Traffic

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Andrew Karam
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UL	24	**19b. TELEPHONE NUMBER (*Include area code)***

Table of Contents

List of Figures

List of Tables

1. Project Objectives

Our project addressed the detection of anomalous behavior in networks. During the funded period we created automatic tools that will detect, through traffic monitoring, anomalous behaviors in computer networks. Because signature techniques cannot detect new forms of attacks, and in the domain of network security, new attacks appear frequently, we focus on designing *adaptive* solutions to quickly detect new(and old) attacks while minimizing the false alarm rate. By adaptive we mean that our approach is designed to detect previously unseen attacks rather than a fixed set of known attacks.

Our approach is to form a model of the normal behavior of a network element and then monitor incoming/outgoing traffic for anomalies. Characterizing normal behavior of a network element requires that we 1) determine where to physically monitor the traffic coming in the element; 2) decide what features of the available data to measure in order to build a scalable, accurate model of the element's normal behavior; and 3) determine how to build a model of normal behavior from the collected data that can then be used for anomaly detection.

Our proposal had three objectives, and we organize the first part of this report with respect to those objectives. In the second part we present some of the experimental results and in the final part, we discuss the utility of our methods for the discovery of covert channels and backdoors.

1.1 Basic Infrastructure

In the first quarter of our funded program, we developed the software infrastructure to collect and monitor data. To monitor the traffic we use snoop. The graduate student on the project has implemented the basic infrastructure in which to run comprehensive experiments.

1.2 Objective 1: Explore how the selection of parameters affects our results

The goals here were to create the infrastructure to automatically select the parameters of our algorithms and to this end, we have implemented an architecture that allows for the automatic selection of the features to use to build a model of normal behavior and this includes the parameter of the window size over which they are computed.

Our implementation of the available features has been completed and we now compute a comprehensive set of features. Note that some of these features are computed over a window of size *n* packets, where *n* is an adjustable parameter of the system. For monitoring network traffic we have code to capture the following features:

$\ast$ Note that this report was written in 2004, but not submitted as the grant was extended to allow for the research on Mouse Movements.

For all protocols: Mean inter-arrival time and standard deviation; Packet length

For IP: Header length, Total Length, ID, Checksum, Percentage of fragmentation, percent of new Source IP, and percent of new Destination IP,

For ICMP: Type, Code, and Checksum.

For TCP: Source port, Destination port, Flags, and Checksum

In addition we have added the capability to collect data by application protocols (i.e., FTP, SSH, HTTP, etc.), and by flow (e.g., TCP traffic from a server to a client of a particular protocol). Note that which features are selected depends on what aspect of normal behavior we are trying to model.

We ran a comprehensive set of experiments to analyze normal behavior of server flows. We experimented with the optimal window size for our aggregate features (see our report from the first quarter). We demonstrated that

- False positive rates of anomaly detection mechanisms monitoring TCP/IP headers can be reduced if we learn a separate model of normal behavior for TCP/IP for each application.

- We can classify a flow as one of a set of applications if we monitor features of their behavior over a window. (See below for experimental results, which expand upon the results reported in the third quarter of our project).

1.3 Objective 2: Expand experimental testbed

Our goals here were to expand the experimental testbed of attacks and run comprehensive experiments. In the first quarter of our funded research we set up an off-the Internet experimental testbed in which to install and test new NDoS attacks. Using a group of undergraduate volunteers, we installed and tested the following attacks: Smurf / Fraggle / Papa Smurf, mstream, TFN, TFN2K and Toast. In addition, we also examined the MIT Lincoln Lab's data and wrote code to allow our methods to be applied to these datasets.

In the second and third quarters we expanded the capabilities in our testbed to analyze traffic by server flow. Previously we partitioned traffic at the transport layer into TCP / UDP / ICMP / IP. We have added the capability to analyze application protocol traffic and further we created the tools to analyze traffic at the flow level (i.e., an application protocol for a particular host).

1.4 Objective 3: Include other network elements

Our preliminary work focused on a Web server and a file server. In the third and fourth quarter of this project, another graduate student, set up the infrastructure to attack and monitor an OSPF network using the the simulation tool OPNet. This allowed us to study the affect of various attacks against routers. Our long-term goal for the this area is to determine the efficacy of anomaly detection in a network at the OSPF level. In particular by watching the link state advertisements. We are now poised to begin experimenting with both signature and anomaly-based techniques for detecting such attacks. We report the results of our intial experiments in the next section.

2. Experimental Results: Investigating the Impact of OSPF Attacks

In our experiments, we used OPNet to first investigate the impact of various types of OSPF attacks and second, to evaluate a very simple rule-based intrusion detection system. Our assumption is that a router has been compromised and is therefore able to send out spoofed Link State Advertisements (LSAs) about other routers in the same domain. In addition, the compromised router can send any information it wants about itself.

In our experiments, the simulated network consisted of 23 routers, 20 servers and 20 LANs. Each LAN was composed of 15 hosts which were HTTP and FTP clients. The clients generated random HTTP and FTP connections to the servers. The connections between the clients and the servers were dynamic. That is, Client A communicating with Server B at time t0 may communicate with another server at time t1.

To investigate the impact of OSPF attacks, we launched several attacks at one of two different locations: a backbone router or an edge router. We discovered that some OSPF attacks could have serious domain-wide impact on the performance of the network. For example, when under an OSPF sink attack, the throughput of the network would drop dramatically due to packet loss. An OSPF sink attack is a domain-wide attack in which the compromised router makes itself a sink by sending out bad LSAs to claim that it has zero cost to everywhere. In addition to the OSPF sink attack, we launched a OSPF MaxAge attack, in which the compromised router sends out bad LSAs about other routers with Age equal to MaxAge. When a router receives a MaxAge LSA, it removes all the links to the sender if that LSA. Therefore, when under the OSPF MaxAge attack, the throughput of the network would drop but not at the same rate as that of the OSPF sink attack.

We also observed that some of the attacks were not noticeable by looking at bandwidth and hence would avoid detection by an administrator. But these attacks were unsuccessful not because of their nature but because the attacker had not yet found out the exact value for the cost that he or she should use to create a loop or a sink. Therefore, while the attacker searches for the cost by trying various values and launching an attack, the impact of such attacks is negligible. But we would still want to know when these unsuccessful attacks are occurring, because they may be an indication that a router has been compromised.

Because there are many different variants of OSPF attacks, we classify the attacks into three groups and then create several general rules. Note that for each class they can be further subdivided by whether the attacker sends the LSAs during the expected update time or not.[1]

1. Sending out bad LSAs about other routers with current sequence number.

2. Sending outbad LSAs about other routers with a sequence number greater than the current sequence number.

3. Sending out bad LSAs about the compromised router itself.

After our initial experiments, we created a simple rule-based IDS that we placed at a backbone router. We then re-launched our set of attacks in order to evaluate the efficacy of our rule-based approach. We discovered that our IDS could detect all attacks except 1) when the compromised router sends out bad information about itself at the expected updating period and 2) when the compromised router sends out bad information about other routers with the current LSA sequence number. Note that for some configurations of attacker and victim router our rules can detect the second case. However missing this second case is less damaging than the first because the impact of this attack is limited. Routers that have already received a correct LSA with current sequence number will not accept the bad LSA with the same sequence number. Therefore, only the routers that receive the bad LSA before the correct LSA would be affected by the attack. Currently the rules of our IDS are:

1. When the IDS detects a new link / router or a down link / router, it will check with the administrator's database to verify if the topology change is expected, which prevents an attacker from adding or dropping link or router.

2. The IDS should only receive LSAs during the expected updating period.

3. The difference between the sequence number of two consecutive LSAs from the same router should be less than or equal to one (Together with Rule 2, this should detect most attacks.)

4. The difference between the age of the received LSA and the database's age of the same originator should be less than *MaxAgeDiff*

5. The change of cost of a link of an originator should be less than N%, where N is a parameter that is configured by the administrator.

6. Information contained in LSAs with the same sequence number should be the same.

[1] Each router is expected to send out LSAs every 30 minutes in our simulation

3. Experimental Results: Classification of Server Flows

Understanding the nature of the information flowing into and out of a system or network is fundamental to determining if there is adherence to a usage policy. Traditional methods of determining traffic type rely on the port label carried in the packet header. This method can fail, however, in the presence of proxy servers that re-map port numbers or host services that have been compromised to act as backdoors or covert channels.

Our experiments were designed to investigate whether we can classify server type based on features of behavior. We present experimental results with learning aggregate flows and by-host flows. The key issue in the behavioral authentication of server flows is what characteristics or *features* of the traffic should be monitored. In environments where there are concerns about user privacy, or where encryption is used to hide the data carried in network packets, we cannot rely on the contents of the payload as a source of features. Rather, we examine the packet header and the operational characteristics of the traffic itself to define our feature set.

For the purposes of our analysis and experiments, we focused on the HTTP, FTP, Telnet, SMTP, and SSH application protocols. These protocols are well understood, stable, widely implemented, and represent the vast majority of user traffic.

Based on our initial observations, we concluded that features based on the TCP state flags (URG-Urgent, ACK -Acknowledgment, PSH -Push, RST -Reset, SYN -Synchronize, and FIN -Finish) can operationally differentiate server flow behavior. For example, HTTP traffic generally contains far fewer packets with the PSH flag than does Telnet traffic. Specifically, for each of the flags, we calculate the percentage of packets in a window of size *n* packets with that flag set. In addition to these six features we calculate the mean inter-arrival time and the mean packet length for the window of *n* packets. During monitoring, these features are used by the classification method to determine whether the previous n packets match the learned behavior of the server flows.

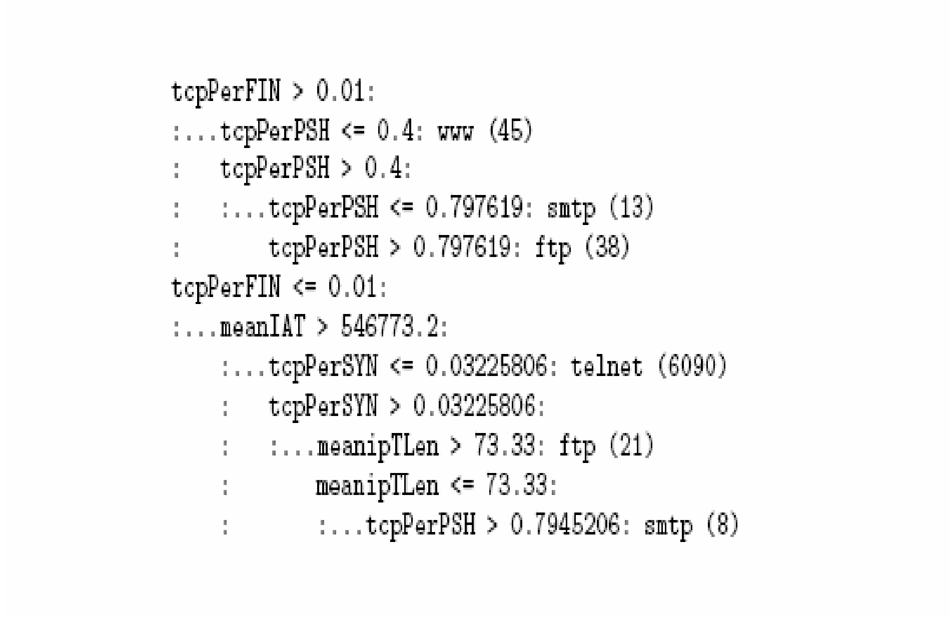
To perform the classification, we chose to use the C5.0 decision tree algorithm – a widely used and tested implementation. Here we provide only the key aspects of the algorithm related to decision tree estimation, particularly as it pertains to feature selection. The most important element of the decision tree estimation algorithm is the method used to estimate splits at each internal node of the tree. To do this, C5.0 uses a metric called the *information gain ratio* that measures the reduction in entropy in the data produced by a split. In this framework, the test at each node within a tree is selected based on splits of the training data that maximize the reduction in entropy of the descendant nodes. Using this criteria, the training data is recursively split such that the gain ratio is maximized at each node of the tree. This procedure continues until each leaf node contains only examples of a single class or no gain in information is given by further testing. The result is often a very large, complex tree that over fits the training data. If the training data contains errors, then over fitting the tree to the data in this manner can lead to poor performance on unseen data. Therefore, the tree must be pruned back to reduce classification errors when data outside of the training set are to be classified. To address this problem C5.0 uses error-based pruning.

3.1 Data Sources

The first data set chosen for our experiments is the 1999 MIT Lincoln Labs Intrusion Detection Evaluation Datasets. Although created for a specific evaluation exercise, these datasets have subsequently been widely used for research into other later intrusion detection systems not part of the original evaluation.

The data represent five weeks of simulated network traffic from a fictional Air Force base. Weeks one through three constitute the *training* data used by anomaly-based intrusion detection systems to model behavior. The data in week one and week three are attack-free. There are five network trace files for each week – one for each business day representing network usage from approximately 8:00 AM to 5:00 PM. Each file is in libpcap format (readable with tcpdump),then compressed using gzip. On average, each week consists of roughly 1 GB of compressed data representing 22 million network packets. We used data from week one in our training sets and data from week three in our test sets. Note that we do not use the attack data, since our purpose is to evaluate whether we can classify server behavior – not whether we can detect intrusions.

In addition to the Lincoln Labs data, we include experiments using data obtained from our own network. The purpose here is to test the applicability of our method on “real world” network traffic. In particular, we are interested in classifying traffic from some of the newer peer-to-peer file sharing protocols –something that the Lincoln Labs data sets do not contain. Some concerns have been raised about the artificial nature of the Lincoln Labs data, and thus an additional objective was to identify any marked differences between experiments with these two data sets.



```
tcpPerFIN > 0.01:
:...tcpPerPSH <= 0.4: www (45)
:   tcpPerPSH > 0.4:
:   :...tcpPerPSH <= 0.797619: smtp (13)
:       tcpPerPSH > 0.797619: ftp (38)
tcpPerFIN <= 0.01:
:...meanIAT > 546773.2:
    :...tcpPerSYN <= 0.03225806: telnet (6090)
    :   tcpPerSYN > 0.03225806:
    :   :...meanipTLen > 73.33: ftp (21)
    :       meanipTLen <= 73.33:
    :       :...tcpPerPSH > 0.7945206: smtp (8)
```

Figure 1: Portion of a decision tree generated by C5.0.

3.2 Aggregate Server Flow Model

Our first experiment was designed to determine the extent to which FTP, SSH, Telnet, SMTP, and HTTP traffic can be differentiated using a decision tree classifier. We used the data from week one of the Lincoln Labs data to build our training dataset. The set was created by first randomly selecting fifty server flows for each of the five protocols. Each server flow consists of the packets from a server to a particular client host/port. The largest flow contained roughly 37,000 packets, and the smallest flow contained 5 packets. The 250 flows represented a total of approximately 290,000 packets. We refer to this as an *aggregate model* because the collection of flows came from many different servers.

The fact that this data is certified as attack-free meant that we could have confidence in the port numbers as indicative of the type of traffic. We used the server port to label each of flows in the training set. Each server flow was then used to generate data observations based on our feature set. The result is a data set consisting of approximately 290,000 thousand labeled observations. We repeated this process for each of seven packet window sizes. The window size is an upper bound on the number of packets used to compute the means and percentages. If an individual flow contains fewer packets than the packet window size, the number of available packets is used to calculate each observation.

Each of the seven training sets was then used to build a decision tree using C5.0. We constructed test sets in the same manner – fifty server flows from each protocol were randomly selected from week three of the Lincoln Labs data. These were then passed to our feature extraction algorithm using the same seven window sizes.

Before describing how a tree is used to classify a flow, we give an example of a portion of a decision tree generated by C5.0 in Figure 1. In this example, the root node tests the percentage of packets in the packet window with the FIN flag set (tcpPerFIN). If this percentage exceeds1%, a test is made on the percentage of packets with the PSH flag set (tcpPerPSH). If this value is less than or equal to 40%, the observation is classified as “www”, indicating HTTP traffic. The numbers in parenthesis indicate the number of training observations classified with this leaf node. Other tests can be seen involving the mean inter-arrival time (meanIAT) and mean packet length (meanIPTLen).

During testing, the class label for a given flow was calculated by summing the confidence values for each observation in the flow. The class with the highest total confidence was assigned to that flow. The classification results are shown in Table 1. For each of seven window sizes, we report the percentage of correctly classified server flows out of the set of fifty flows for each protocol. As can be seen in the table, the classification accuracy ranges from 82% to 100%.

Table 1: Classification accuracy of the aggregate model decision trees on unseen individual server flows. Each value represents the percentage of correctly classified flows out of the fifty flows for each protocol

Window Size	FTP	SSH	Telenet	SMTP	WWW
1000	100%	88%	94%	82%	100%
500	100%	96%	94%	86%	100%
200	98%	96%	96%	84%	98%
100	100%	96%	96%	86%	100%
50	98%	96%	96%	82%	100%
20	100%	98%	98%	82%	98%
10	100%	100%	100%	82%	98%

In general, the classification accuracy was lower for SMTP server flows than for other protocols. We examined the misclassified flows in more detail and discovered that these flows were generally 24 times longer than correctly classified flows. Longer SMTP server flows represented longer periods of interaction, and thus contain increasing numbers of observations classified as Telnet or FTP. In these few cases, our feature set is not adequate for discriminating between the behaviors of these flows.

It is more desirable to use a smaller window size because this decreases the time to detect that a service is behaving abnormally. Indeed for SSH we see that too large a packet window size (1000) hurts classification accuracy. For FTP, SSH and Telnet, a window size as small as ten packets achieves 100% classification accuracy.

Because the proposed method would be used to monitor traffic in real time, we did a rough calculation of classification time. The average length of time used by C5.0 to classify an entire flow was 70mS.[2] Training is done offline so computation time is of lesser importance, but note that the average length of time used by C5.0 to create each decision tree was 22 seconds. Finally, we need to address the storage requirements for maintaining a window of *n* values to compute the value of each of the features. We can approximate the value created by storing all *n* values by retaining only the mean for each feature, μF_i and using the following update rule for each new packet:

$$\frac{(n-1)\mu F_i + newF_i}{n}$$

In future work we will investigate whether this technique significantly degrades performance.

We conclude from our experimental results that the behavior of server flows for the five protocols can be differentiated using a decision tree classifier built on aggregate flows.

[2] The hardware platform used for building the decision trees and classifying observations was a 500Mhz Dual Pentium III PC with 772MB of RAM running Red Hat Linux (kernel version 2.4.18).

3.3 Host-Specific Models

Our second experiment addresses whether creating models for specific hosts provides better performance than the aggregate model. There are three advantages to using host-specific models:

1. By creating models for individual server flows, we can monitor these flows for changes in behavior.

Table 2: Number of flows used for each protocol in training and test sets for each host model

Host	Training Flows	Test Flows
172.16.112.100	20	20
172.16.112.50	30	25
172.16.113.50	35	23
172.16.114.50	10	20
197.218.177.69	25	35

Table 3: Classification accuracy of host model decision trees on unseen server flows. Each row reports the host address and the percentage of correctly classified flows for each protocol. Fields with a "–" indicate there was no traffic of this protocol type for this host.

Host	FTP	SSH	Telenet	SMTP	WWW
172.16.112.100	95%	-	100%	90%	100%
172.16.112.50	92%	100%	84%	100%	-
172.16.113.50	100%	-	100%	100%	-
172.16.114.50	100%	95%	100%	95%	95%
197.218.177.69	100%	-	100%	100%	-

2. A host-specific model can capture the implementation subtleties of a particular service running on a host. This resolution is missing in the aggregate model consisting of many server flows.

3. The training examples in an aggregate model will be dominated by the server generating the most traffic. This may dilute examples from other servers. The host-specific model solves this problem.

We first identified a set of hosts in the Lincoln Labs data that each ran three or more server protocols. Training data for each host was collected by randomly selecting server flows from week one for each of the protocols running on these hosts. The number of flows used in each model was chosen such that each protocol was represented by the same number of flows. Table 2 lists the number of training and test flows per host.

Based on our results using the aggregate models, we chose a packet window size of 100 for generating observations. The selection was driven by the fact that SMTP accuracy was greatest using this window size with the aggregate models, and other protocol classifications accuracies were between 96% and 100%. We then trained a decision tree for each host that could be used to differentiate the server flows coming from that host. Test data was collected from week three in the same manner as the training data.

The results in Table 3 indicate that, in general, the host specific models achieve approximately the same classification accuracy as the aggregate models. One difference observed is that classification accuracy varies by protocol. For example, the classification accuracy of Telnet flows for host *172.16.112.50* is 84% where as the classification of Telnet flows in the aggregate models averaged 96.2%. Examination of the packets in the misclassified Telnet flows revealed an interesting phenomenon. We often observed large time gaps between packets. The time gaps indicate lapses in user activity where the Telnet server is not echoing characters or supplying responses to commands. In our framework, a single large gap can radically alter the values for the mean inter-arrival time of packets, thus resulting in misclassification of the subsequent observations. We refer to this as the *Water Cooler Effect* – the user temporarily leaves the interactive session, then resumes a short while later. We are investigating the sensitivity of our classifiers to this effect. One possible solution would be to subdivide flows based on sometime gap threshold and use the interactive sub-flows to build our classifiers.

3.4 Models from Real Network Traffic

In this section we present experiments with real network traffic. We collected a number of server flows using the protocols described. We augmented this set to include flows from hosts acting as Kazaa servers. Kazaa is a peer-to-peer file sharing system that is growing in popularity. Peer-to peer network traffic was not part of the Lincoln Labs dataset.

Our goal was to determine if there was a significant difference in classification accuracy when using synthetic versus real traffic. We observed classification accuracies by protocol ranging from 85% to 100% for both the aggregate and host models. The peer-to-peer traffic was classified correctly for 100% of the unseen flows. This is an especially interesting result because Kazaa flows carry a port label that is user-defined. Thus, we are able to correctly classify peer-to-peer flows behaviorally – without the use of the port number. These results indicate that our classification method is effective for real network traffic. The range of accuracies match those observed with the synthetic data. Thus, we can identify no appreciable difference in the per-flow behavior in the synthetic Lincoln Labs data versus those in real network traffic.

3.5 Conclusions

We have presented a novel approach for defining a set of features to model operational behavior of server flow traffic. We demonstrated through the use of the C5.0 decision tree algorithm that our features can differentiate the behavior of server protocols with an accuracy of 82% to 100%. We illustrate empirically that aggregate models can classify an unseen server flow as belonging to a family of previously seen flows, and that host models can determine whether flows from a given server match the behavior of previously seen flows from that server. These classifiers can augment traditional intrusion detection systems to detect artifacts of successful attacks. Our techniques of classification are independent of packet labelings and are thus immune to techniques that modify port numbers to conceal activity.

This technique provides a valuable tool for use in identifying the true nature of a given server server flow. It is independent of packet labelings and is thus immune to techniques that modify port numbers to conceal activity.

4. Published Papers and Presentation

This was a one year grant, that received a no-cost extension. It was then expanded to include a sub-project funded by the NSA on User Re-Authentication via Mouse Behavior. The results from that sub-project can be found in a separate final-report also submitted to AFRL. This grant supported one PhD student, James Early, who is currently a visiting professor in the Department of Computer Science at Purdue University. The papers and presentations from this grant are:

Publications:

- Early, J., Brodley, C. E. and Rosenberg, C., "Behavioral Authentication of Server Flows," in the *Proceedings of the Nineteenth Annual Computer Security Applications Conference,* December 2003, Las Vegas, pp. 49-55.

- Early, J. P. and Brodley, C. E. "Behavioral features for network anomaly detection," in M. Maloof, (Ed.) Machine Learning and Data Mining for Computer Security: Methods and Applications, Springer, 2005, pp. 107-124.

- Early, J.P., "Feature Extraction to Describe Attribute Behavior," Ph.D. Thesis, Department of Computer Science, Purdue University, 2005.

Presentations:

- "Automatic Detection of Anomalous Behavior in Networks," Air Force Research Laboratory, Rome, NY, November, 2002, C. Brodley.

- "Using Statistics to Detect and Thwart Denial of Service Attacks," Interface-2003, Salt Lake City, March 2003, C. Brodley.

- "Behavioral Authentication of Server Flows," Nineteenth Annual Computer Security Applications Conference, December 2003, Las Vegas, J. Early.

- "Behavioral Authentication for Computer Security," C. Brodley

  – Department of Computer Science, Brandeis University, Waltham, MA, October 2003

  – Department of Computer Science, University of Massachusetts, Amherst, MA, October 2004

Appendix A
Final Report - User Re-authentication via Mouse Behavior

Dr. C. E. Brodley, Principal Investigator

Department of Computer Science
Tufts University
Medford, MA 02144
brodley@cs.tufts.edu
617-627-3652

July 31, 2006

1 Project Objectives

Our long-term objective is to provide ways to learn and adapt a model of human behavior to detect anomalies. An anomaly could signal misuse, an intrusion or insider threat. The goal of this project was to evaluate whether we can model user-behavior through mouse movements. Although the original proposal was restricted to mouse movements, we expanded the scope of the research to include several other biometric sources. In particular we monitor each user's keystroke, mouse and GUI behavior to determine if he/she is a valid user or an imposter. In this final report, we summarize our research findings.

2 Task 1: Collection of a Large Scale Database

During the course of the project, we collected two distinct data sets. The first data set was obtained by a group of 61 volunteers who were given a reading assignment followed by a set of twenty questions about the material they just read. Users were specifically instructed to read the assignment on the screen as opposed to reading it from a printout in order for us to record as many mouse movements and GUI changes as possible while they navigated between the reading material and a text editor. They were also given a set of web pages to look at and answer yet another set of questions. The average number of hours to complete the assignment was 4.10 hours.

The second dataset was collected by a group of 47 volunteers who were asked to fill out an electronic copy of a travel expense form (one page in length) word-by-word from a template. Typing mistakes were allowed. Users were instructed to use their mouse device when going from one field of the form to the next (instead of the "Tab" key) in order for us to record as many mouse movements as possible. The entire process lasted 9.44 minutes on average.

In our experiments we used a ten-fold cross validation. In each of the ten runs 90% of the user data set was used for training and the remaining 10% was used for testing. Please note that a different 10% of the user data set was used for testing in each run. The results reported were averaged over the inner eight cross-validation runs. We decided to disregard results obtained from the first and last cross-validation run because our volunteers used this time to familiarize themselves with the data collection software and to wrap-up the data collection process, respectively.

3 Task 2: Develop Features for Discrimination of Users

In this section we describe the full set of extracted features. Note that feature selection is an integral part of our classification scheme, and that the classifiers induced from the data had far fewer features than the full set.

Each user dataset contains keystroke, mouse and GUI data. The data points are recorded in the order in which they are induced by a user. For each data point we record the time, the X and Y screen coordinates and the application in which the data point occurred. We compute features by examining a *window of N* data points at a time. Ideally, the parameter N is customized for each user, but for the experiments in this paper we set N at five hundred.[1]

We collect keystroke events for every key on a general–purpose keyboard (see Figure **??**). This includes the "regular" keys (i.e., letters of the alphabet and numbers), "function" keys (e.g., F1–F12), "control" keys (e.g., Control, Alt and Delete), "mouse" keys (i.e., keys used by expert users to navigate through a computer system without the use of a mouse device, such are Page Up/Down, Tab, arrow keys, etc.) and "other" keys (e.g., Pause/Break, PrtSc/SysRq, etc.). We extract keystroke features by computing the *n-graph* duration[2] between consecutive keystrokes where $n \in [1, 8]$.

We collect the following mouse events: left/right clicks and double clicks, mouse wheel movements and non-client (NC) area mouse movements[3]. Due to an intractably high volume of client-area mouse movements[4] we rate limit them by recording a new mouse movement data point every 100 milliseconds *if and only if* the on-screen cursor moved in the meantime. Thus, our mouse data is composed of mouse events and mouse movements. For every two consecutive data points we compute the following eleven features: 1) distance, 2) speed, 3) angle of orientation and 4) *n-graph* duration where $n \in [1, 8]$.

To obtain GUI data points we record the following events: "window" events (i.e., scroll bar, minimize, maximize, restore, move, etc.), "control" events (i.e., application and process control, open/close, etc.), "menu" events (e.g., open, select, navigate, close), "item," "icon," "dialog," "query," and "combo box" events (e.g., open/close, select, move, resize, etc.) and "miscellaneous" events (e.g., power up/down, language change, background color change, etc.). Some of these events are purely temporal in nature and some are both temporal and spatial. We extract features for *all* temporal GUI events by computing the *n-graph* duration between consecutive events where $n \in [1, 8]$. For spatial GUI events we compute the 1) distance, 2) speed and 3) angle of orientation between every two consecutive GUI points.

Finally, to complete the feature extraction process we compute the mean, the standard deviation and the third moment of the keystroke, mouse and GUI features over a *window of N* data points. To summarize, our feature space contains the following:

1. For each type of data points (there are eight of them for keystrokes, eight for mouse and twelve for GUI, and for the entire window of N data points we compute three statistical measurements (i.e., the mean, the standard deviation and the third moment) of eight keystroke features (i.e., n-graph duration with $n \in [1, 8]$), eleven mouse features (i.e., distance, speed, angle of orientation and n-graph duration with $n \in [1, 8]$), eight temporal GUI features (i.e., n-graph duration with $n \in [1, 8]$) and three spatial GUI features (i.e., distance, speed, angle

[1] $N = 500$ is equivalent to a fifty second time period on average.

[2] The n-graph duration is defined as the elapsed time between the first and the nth data point [**?**].

[3] Non-client area of a window is considered to be the portion of the window where toolbars and menu are located.

[4] Client-area mouse movements occur over a hundred times per second even when user has only touched the mouse device and has not moved it even by a pixel.

of orientation). This generates $8 * 3 * 8 = 192$ keystroke features, $8 * 3 * 11 = 264$ mouse features, $12 * 3 * 8 = 288$ temporal GUI features, $6 * 3 * 3 = 54$ spatial GUI features and $1 * 3 * 11 = 33$ features for the entire window of N data points.

2. For each type of mouse and spatial GUI data points and for the entire window of N data points we compute three statistical measurements of X and Y cursor coordinates, thereby gaining an insight into which part of the computer screen each user was most or least active in. This generates $8 * 3 * 2 = 48$ mouse features, $6 * 3 * 2 = 36$ spatial GUI features and $1 * 3 * 2 = 6$ features for the entire window of N data points.

3. Over a *window of* N data points we count the number of points of each type. This generates another 28 features. We also count the number of occurrence of each alphabet letter and each numeral thereby obtaining 26 alphabet features (i.e., A(a)–Z(z)) and ten (i.e., 0–9) numeric features.

4 Task 3: Perform Extensive Experimentation

We begin this section with the description of the learning algorithm used for classification and the evaluation method used in the experiments. We then present a series of experiments designed to investigate the applicability of biometric dynamics in user re-authentication. Note that we have a far more comprehensive set of experiments in our papers.

4.1 Classification

We apply a supervised learning algorithm to determine whether we can discriminate the users. We assume a closed-setting scenario in which data can be readily obtained from all of the employees and intruders are likely to come from "within". We use the decision tree algorithm C5.0 (without boosting) to build a clasifier that is then used to classify new instances as either belonging or NOT belonging to a valid user. We chose to use decision trees because they provide a comprehensible representation of their classification decisions. Although techniques such as boosting or support vector machines might obtain slightly higher classification accuracy, they will not greatly impact the results and further they obscure the decision making process. Finally, note that decision tree algorithms perform automatic feature selection.

4.2 Evaluation Methodology

We implement two evaluation metrics. We begin by testing one feature vector instance at a time. If the instance belongs to the profile of a valid user, we service the user's request. If, on the other hand, the instance does not belong to the profile of a valid user, we do one or all of the following: alert the system administrator, ask the user to authenticate again and close the current login session. Note that such a simplistic scheme may induce a high rate of false alarms. To address this issue we implemented a smoothing filter. We look at a window of $n \in [1 - 11]$ feature vector instances at a time and if $m \in \{0, n\}$ of those instances belong to a valid user, we service the user's request, otherwise, we raise an alarm. Note that both n and m are user specific and empirically derived on the training datasets by minimizing the unweighted sum of false error rates. We define a *cluster* of false alarms to be the sequence of consecutive false alarms and we count each sequence as a single false alarm. Our argument for this choice is that when an alarm is raised for the first time, the bell is turned on and kept on until it is serviced by a system administrator. Consecutive alarms

Performance Measures	FP Rate		FB Rate		FN Rate		Error Rate		Bell Count	
STATISTIC	AVG	STD	AVG	STD	AVG	STD	AVG	STD	AVG	STD
Basic	5.87%	3.50	1.40%	0.62	5.87%	1.93	4.98%	1.56	2.21	1.02
Smoothing	1.71%	1.40	0.33%	0.27	10.58%	4.43	5.44%	1.98	0.50	0.42

Table 1: **The average and the standard deviation values of FP, FB, FN and Error Rates and Bell Count over all 61 users for pairwise discrimination.**

Performance Measures	FP Rate		FB Rate		FN Rate		Error Rate		Bell Count	
STATISTIC	AVG	STD	AVG	STD	AVG	STD	AVG	STD	AVG	STD
Basic	45.70%	17.30	6.28%	1.70	0.54%	0.27	1.21%	0.60	10.85	5.76
Smoothing	21.11%	15.97	1.85%	0.94	11.75%	29.40	11.95%	29.01	3.15	1.82

Table 2: **The average and the standard deviation values of FP, FB, FN and Error Rates and the Bell Count over all 61 users for binary classifiers.**

are observed as being a part of the original bell. By counting the number of *bells* and dividing this count by the number of valid user's instances we obtain the *false bell rate* (i.e., FB Rate).

4.3 Experiment I: Pairwise Discrimination

The result of this experiment illustrated the strength of a biometric classifier as a user re-authentication tool. We wanted to answer the question of whether there is a signal of "normalcy" per user and if so, to what extent. To this end, we built a binary classifier to perform pairwise discrimination between every two users thereby generating a symmetric 61 by 61 matrix. Table 1 summarizes the results averaged over all 61 users. The results obtained confirm that there is a signal of "normalcy" per user. Small standard deviation values of FP, FB, FN and Error rates and Bell Count suggest good scalability of the pairwise biometric classifier. Nonetheless, the most important outcome of this experiment was an insight into which features capture user behavior best on a per–user–basis. By examining the decision trees from the pairwise experiment we were able to produce a subset of most relevant features. The feature subset is *user specific* and the feature subset for user A has a union over all pairwise decision trees generated for user A of the ten most *significant*[5] and *unique* features that distinguish user A from some user B where $B \in \{1, N\}$ and B is different from A. In the following experiments we use the individual feature subsets to evaluate the performance of the biometric classifier; i.e., for user A we use feature set A.

4.4 Experiment II: Anomaly Detection

This experiment was designed to test the accuracy of a biometric classifier for each user. In particular to answer whether one can build an accurate model of normal user behavior after seeing the behavior of a valid user A and the behavior of the remaining $N - 1$ users in the role of intruders. Table 2 summarizes results over all 61 users. The smoothing filter scheme lowered FP and FB rates and the Bell Count, but increased FN and Error rates.

We examined the behavior of individual users to determine why some users produced either a high FB or FN rate. We examine their raw data files, their feature vectors and their decision trees and summarize our observations in Table 3 and Table 4. Table 3 shows users with FB rate

[5]The feature significance is determined by its entropy measure.

ID	FB Rate	Elapsed Time	#Mouse Points	#Keystroke Points
4	5.34%	8.88 hrs	**37.4**	**18.41**
18	9.32%	**0.38 hrs**	**31.0**	**2.0**
38	8.67%	**0.87 hrs**	141.6	**1.3**
44	8.11%	**0.90 hrs**	156.9	22.8
AVG	2.50%	4.1 hrs	86.0	28.4

Table 3: **Users with FB rate above 5%. The average values in the table are obtained by averaging over all 61 users.**

ID	FN Rate	Elapsed Time	#Mouse Movements	#Keystroke Points
2	3.73%	3.35 hrs	**37.3**	34.2
8	3.76%	18.43 hrs	44.6	**3.8**
22	3.74%	**2.48 hrs**	**32.2**	**11.2**
28	3.76%	**1.33 hrs**	57.8	63.3
42	3.73%	**0.72 hrs**	81.5	23.1
48	3.75%	**43 min**	188.3	58.6
AVG	1.96%	4.1 hrs	65.5	28.4

Table 4: **Users with FN rate above 3.5%. The average values in the table are obtained by averaging over all 61 users.**

above 5%. The first column of the table shows User ID, second column has the FB rate for the specific user; the third column has the time it took user to complete the data collection process; and the fourth and fifth columns show the average number of all mouse points and all keystroke points, respectively. The last row of the table shows the average values across all 61 users. Table entries printed in bold highlight those values that are approximately one standard deviation above or below the average. Closer examination of Table 3 reveals that the number of mouse points was very low for users 4 and 18. Although these users moved a mouse to complete the assignment, they induced very few mouse events which made it difficult for the classifier to distinguish them from the rest. Similarly, users 4, 18 and 38 had only a few keystroke points. Finally, users 18, 38 and 44 spent less than an hour collecting the data which is one-fourth of the average data collection time. In other words, these users did not have enough data for the biometric classifier to build an accurate profile of their normal behavior. The classifier misidentified them as intruders and this in turn created a high FB rate. We conclude that in order to improve the overall accuracy, we need more raw data per user dataset and this is particularly true as the number of users to be discriminated increases.

Table 4 shows users with FN rate above 3.5%. The first column of the table shows User ID, second column has the user's FN rate, third column has the elapsed time of the data collection; and the fourth and fifth columns show the average number of mouse movements and keystroke points, respectively. Closer examination of Table 4 reveals that users 22, 28, 42 and 48 spent less than two and a half hours collecting the data which is about half the average time. Furthermore, users 2 and 22 had less than two thirds of the average number of mouse movements; and users 8 and 22 had well below the average number of keystroke points. All these factors contributed to the failure of the biometric classifier to generalize well. The classifier overfitted the data and created a *too broad* profile of valid user's behavior. Subsequently, when an intrusion occurred the intruder's behavior was misidentified as that of the valid user and an alarm was not raised. We conclude that the lack of data per user leads to an inaccurate profile and consequently high FB or FN rate.

4.5 Experiment III: Detecting Previously Unseen Intruders

The hypothesis we want to test is whether the biometric classifier is accurate enough to detect even an "unseen" intruder after previously being trained on a large number of "seen" intruders. We investigate the performance of the classifier when all but one randomly selected user data set (let's refer to this data set as U) is seen in the training phase and a profile of normal user behavior for each user is built by observing the valid user's behavior and behavior of the remaining $N-2$ intruders. We use U's data set to compute the FN rate during testing. The average FN rate across all 61 users is 0.98% and in fact, it is zero for 15 users. We conclude that the biometric classifier is accurate enough to detect even a previously unseen intruder (i.e., an unlabeled sample).

5 Conclusions

We take a direct approach to user re-authentication by continuously monitoring user's mouse movements, keystroke dynamics and GUI changes in a syntactic sense. We purposefully avoid the keystroke semantics to increase the efficiency of our system. Namely, once the model of normal user behavior (i.e., a binary decision tree) is obtained we need only compute those features that are present in the tree and then classify the current user by traversing the tree. This makes our approach fast and therefore suitable for an on-line environment.

We conducted four experiments to determine the strength and gain insight into the scalability of the biometric classifier. Our results indicate that even a classifier trained over a short period of time can scale well and accurately build a profile of valid user behavior *if and only if* user utilized his/her I/O devices. We conclude that *both* a longer data collection time and more raw data instances per user are needed for the classifier to scale well and build an accurate profile of user behavior. As a final point, our third experiment relaxed the constraints of the closed-setting scenario by demonstrating that the biometric classifier did not need to have previously seen every intruder's data set to accurately detect the intruder *assuming* the classifier had been trained on a large number of seen intruders. These results suggest that as the number of seen intruders increases in the training phase, more unseen intruders (i.e., more unlabeled data) could be detected in practice. In summary, the biometric sources of mouse movements, key strokes and GUI behavior provide a method for authentication of users.

6 Published Papers and Presentation

Publications:

- Pusara, M. and Brodley, C. E., "User Re-authentication via Mouse Movements," *the Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, September 24-26th, 2003.

- Pusara, M. and Brodley, C. E., "User Re-authentication via Mouse Movements," *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, 2004, Washington DC, USA October 29 - 29, 2004

- Pusara, M. and Brodley, C. E., "Analysis of Mouse Dynamics for User Re-authentication," *submitted to ACM Transactions on Information and System Security.*

- Pusara, M. and Brodley, C. E., "Dynamics of Biometric Sources for User Re-authentication," *submitted to ACM Conference on Computer and Communications Security.*

Presentations:

- "User Re-authentication via Mouse Movements," Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection, Washington, D.C. September 24-26th, 2003, Maja Pusara.
- "User Re-authentication via Mouse Movements," 2004 ACM workshop on Visualization and data mining for computer security, 2004, Washington DC, USA October 2004, Maja Pusara.
- "Behavioral Authentication for Computer Security," C. Brodley
  - Department of Computer Science, Brandeis University, Waltham, MA, October 2003
  - Department of Computer Science, University of Massachusetts, Amherst, MA, October 2004